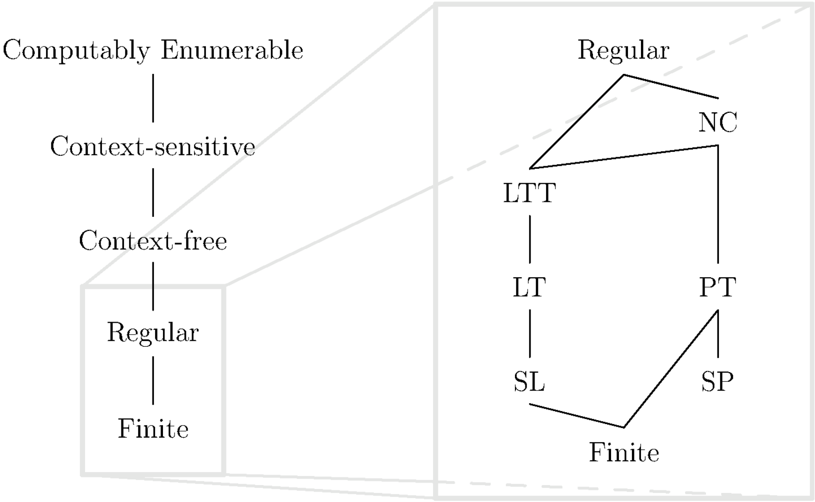
Complexity of Subregular Stringsets

Jeffrey Heinz

March 2, 2021

Computational Complexity



My own interests in subregular complexity...

...began with these observations:

1. Many phonotactic patterns are regular.
2. Many regular patterns are *not* possible phonotactic patterns.

So *which* regular languages constitute possible phonotactic patterns?

Along the way I learned:

1. There is a rich field of study in computer science on *subregular* complexity.
2. There is a rich field of study in computer science on *machine learning* of regular languages and transductions.
3. There are many applications in linguistics, artificial intelligence, robotic planning and control, model checking, ...

Measuring Complexity

One idea is that a regular language L_1 is *more complex* than L_2 if

1. The smallest DFA recognizing L_1 is larger than the smallest DFA recognizing L_2.
2. The smallest regular expression recognizing L_1 is larger than the smallest regular expression recognizing L_2.

In contrast to these *intensional* measures

1. The subregular classes we study today will provide complexity measures *independent* of the size of such representations.

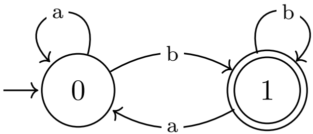
Intuitions?

Consider the four sets of strings below. Any intutions about which set of strings is more or less complex than any others?

- Strings must end with `b`.
- Strings have odd many `b`s.
- Strings have at least one `b`.
- Strings have exactly one `b`.

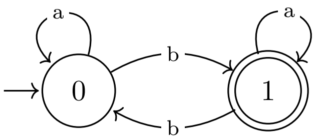
Comparing "final-b" with "odd-b"

Strings must end with `b`.



`(a + b)*b`

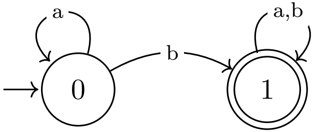
Strings have odd many `b`s.



`a*ba*(a*ba*ba*)*`

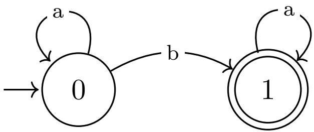
Comparing "at least one b" with "exactly one b"

Strings have at least one `b`.



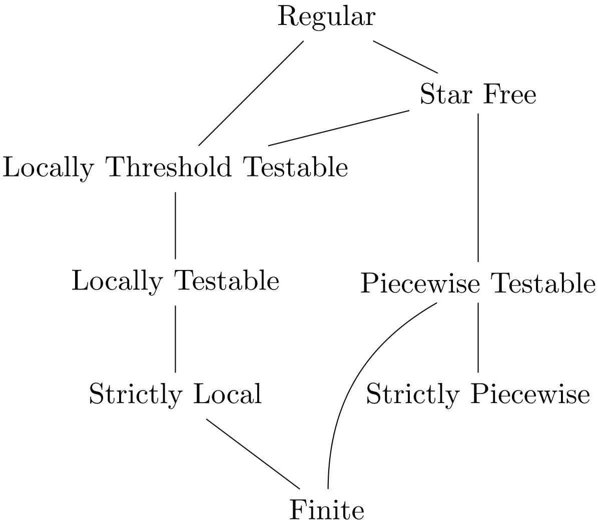
`a*b(a+b)*`

Strings have exactly one `b`.

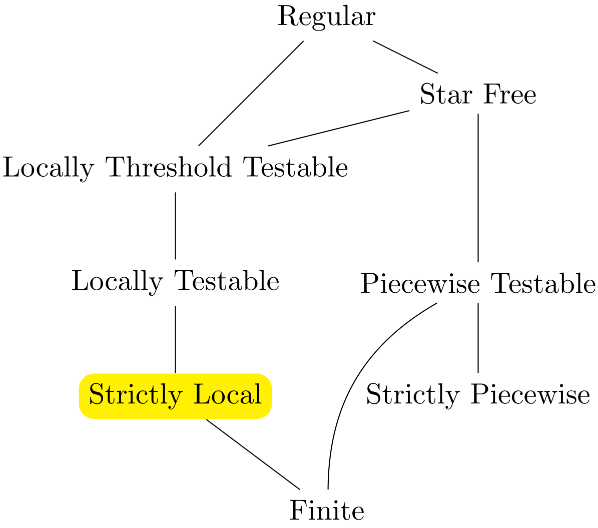


`a*ba*`

Subregular Complexity



Subregular Complexity



Factors

The notion of *substring*, also called *factor*, is key to understanding the "local" subregular classes.

k-Factors

- A string u is a k-factor (substring of length k) of a string v if there exist strings x, y such that $v = xuy$ and $|u| = k$. We write $u \sqsubseteq_k v$.
- $\texttt{factor}_k(w)$ equals the set $\{u \mid u \sqsubseteq_k w\}$ if $|w| > k$ and equals $\{w\}$ if $|w| \leq k$.
- $\texttt{factor}_k(S)$ equals the set $\bigcup_{w \in S} \texttt{factor}_k(w)$.

Exercise

Identify the following sets.

1. $\texttt{factor}_2(abcd)$
2. $\texttt{factor}_3(abcd)$
3. $\texttt{factor}_6(abcd)$
4. $\texttt{factor}_2(aaaa)$
5. $\texttt{factor}_2(aaaa, bbbb, caabc)$

Strictly Local Languages

Let $\rtimes$ and $\ltimes$ denote left and right word boundaries.
A language L is Strictly k-Local if there is a finite set of strings $S \subseteq \texttt{factor}_k\big(\{\rtimes\} \cdot \Sigma^* \cdot \{\ltimes\}\big)$ such that

$$L = \{w \in \Sigma^* \mid \texttt{factor}_k(\rtimes w \ltimes) \subseteq S\}$$

A language L is Strictly Local if there is some k for which it is Strictly k-Local. We sometimes write $[\![\mathrm{S}]\!]$ to denote the extension of S.

Strictly Local Languages

Notes

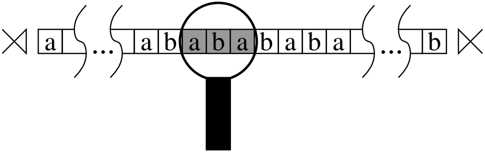
- Elements in S are called **permissible factors**.
- Elements in $\texttt{factor}_k\big(\{\rtimes\} \cdot \Sigma^* \cdot \{\ltimes\}\big)$ not in S are called **forbidden factors**.

Exercises

1. Let $S = \{\rtimes a, ab, ba, a\ltimes\}$. What is $[\![\mathrm{S}]\!]$?
2. For what k is $[\![\mathrm{S}]\!]$ Strictly k-Local?
3. Which factors are forbidden?

Strictly Local Languages: Scanners

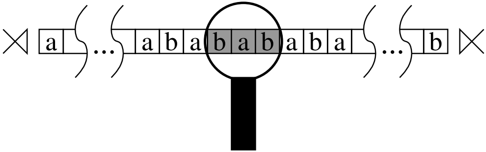
Intuitively, if L is Strictly k-Local, then deciding whether a string w belongs to L simply requires scanning w for forbidden substrings. If any is found w is rejected. If every substring in w is permissible then w is accepted.



(McNaughton and Papert 1971, Rogers and Pullum 2011)

Strictly Local Languages: Scanners

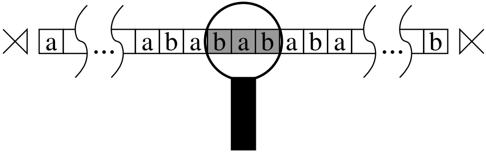
Intuitively, if L is Strictly k-Local, then deciding whether a string w belongs to L simply requires scanning w for forbidden substrings. If any is found w is rejected. If every substring in w is permissible then w is accepted.



(McNaughton and Papert 1971, Rogers and Pullum 2011)

Strictly Local Languages: Scanners

Intuitively, if L is Strictly k-Local, then deciding whether a string w belongs to L simply requires scanning w for forbidden substrings. If any is found w is rejected. If every substring in w is permissible then w is accepted.



(McNaughton and Papert 1971, Rogers and Pullum 2011)

More exercises

Consider the languages below.

(1) Strings end with a `b`.

(2) The second to last symbol in all strings is `b`.

(3) The third to last symbol in all strings is `b`.

For each, one explain why it is Strictly Local. Assume $\Sigma = \{\texttt{a},\texttt{b}\}$.

Theorems

1. $\mathrm{SL}_1 \subsetneq \mathrm{SL}_2 \ldots \mathrm{SL}_k \subsetneq \mathrm{SL}_{k+1} \ldots \subseteq \mathrm{SL}$
2. $\mathrm{FIN} \subsetneq \mathrm{SL}$
3. For each k, $\mathrm{FIN} \not\subseteq \mathrm{SL}_k$
4. For each k, SL_k is closed under intersection, but neither complement nor union.

Grammar-independent Characterization of SL

Suffix Substitution Closure
A language L is Strictly Local iff there is a k such that for all strings $u_1, v_1, u_2, v_2 \in \Sigma^*$ and for all strings x of length $k-1$ whenever $u_1 x v_1, u_2 x v_2 \in L$ then $u_1 x v_2 \in L$.

In a picture

$$\begin{array}{cccc} u_1 & \overbrace{x}^{k-1} & v_1 & \in L \\ u_2 & x & v_2 & \in L \\ \hline u_1 & x & v_2 & \in L \end{array}$$

Proving some languages are not SL

The SSC helps us prove languages are not Strictly k-Local and even not Strictly Local for any k.

- To show L is not SL_k, find u_1, v_1, u_2, v_2 and x of length $k-1$ such that

In a picture

$$
\begin{array}{cccc}
 & \overbrace{}^{k-1} & & \\
u_1 & x & v_1 & \in L \\
u_2 & x & v_2 & \in L \\
\hline
u_1 & x & v_2 & \notin L
\end{array}
$$

- To show L is not SL, find u_1, v_1, u_2, v_2 and x for each k!

Exercises

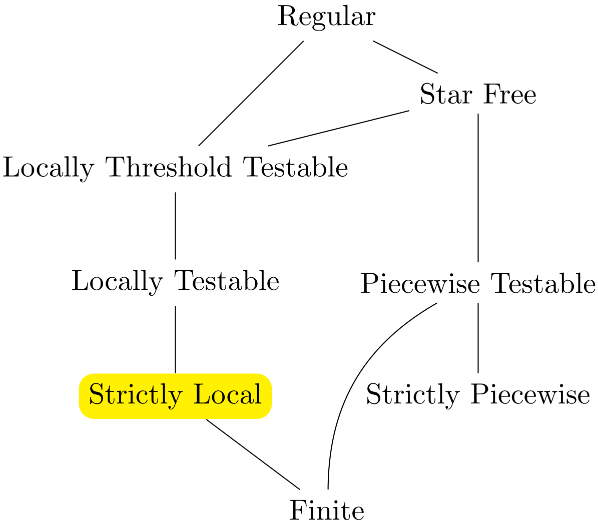
Prove the following languages are not SL for any k.

(4) Strings contain at least one `b`.

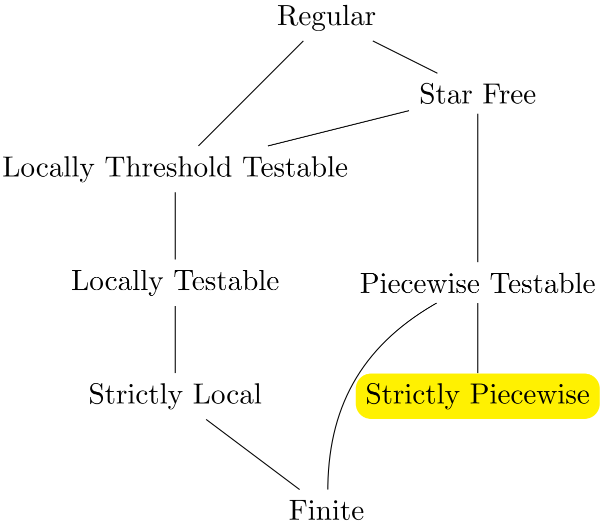
(5) Strings contain at most one `b`.

(10) Strings contain an `a` between every pair of `b`s.

Subregular Complexity



Subregular Complexity



Subsequences

The notion of *subsequence* is key to understanding the "piecewise" subregular classes.

k-Subsequences

- A string u is a subsequence of a string $v = \sigma_1\sigma_2\ldots\sigma_n$ if

  $$v \in \Sigma^*\sigma_1\Sigma^*\sigma_2\Sigma^*\ldots\Sigma^*\sigma_n\Sigma^* \ .$$

  We write $u \preceq v$.
- $\texttt{subseq}_k(w)$ equals the set $\{u \mid u \preceq_k w, |u| \leq k\}$.
- $\texttt{subseq}_k(S)$ equals the set $\bigcup_{w\in S} \preceq_k (w)$.

Exercise

Identify the following sets.

1. $\texttt{subseq}_2(abcd)$
2. $\texttt{subseq}_3(abcd)$
3. $\texttt{subseq}_2(aaaa, bbbb, caabc)$

Strictly Piecewise Languages

A language L is Strictly k-Piecewise if there is a finite set of strings $S \subseteq \texttt{subseq}_k\big(\Sigma^*\}\big)$ such that

$$L = \{w \in \Sigma^* \mid \texttt{subseq}_k(w) \subseteq S\}$$

A language L is Strictly Piecewise if there is some k for which it is Strictly k-Piecewise. We use $[\![\mathrm{S}]\!]$ to denote the extension of S.

Strictly Piecewise Languages

Notes

- Elements in S are called **permissible subsequences**.
- Elements in $\texttt{subseq}_k(\Sigma^*)$ not in S are called **forbidden subsequences**.

Exercises

1. Let $S = \{\varepsilon, a, b, ab, ba\}$. What is $[\![\mathrm{S}]\!]$?
2. For what k is $[\![\mathrm{S}]\!]$ Strictly k-Piecewise?
3. Which subsequences are forbidden?

More exercises

1. Recall this language mentioned earlier.

   (5) Strings contain at most one `b`.

   Explain why it is Strictly 2-Piecewise. Assume $\Sigma =$ {`a`,`b`}.

2. Recall Samala.

possible Samala words	impossible Samala words
ʃtojonowonowaʃ	stojonowonowaʃ
stojonowonowas	ʃtojonowonowas
pistonoskiwat	pisotonoʃikiwat
sanisotonoskiwas	ʃanipisotonoʃikiwas

   Assuming $\Sigma =$ {`s`,`S`,`t`,`o`}, what are the forbidden subsequences?

Strictly Piecewise Languages

Intuitively, if L is Strictly k-Piecewise, then deciding whether a string w belongs to L simply requires checking its k-subsequences for forbidden ones. If any is found w is rejected. If every subsequence in w is permissible then w is accepted.

(Rogers et al. 2010)

Infinite Hierarchy of SP classes

Theorems

1. $\mathrm{SP}_1 \subsetneq \mathrm{SP}_2 \ldots \mathrm{SP}_k \subsetneq \mathrm{SP}_{k+1} \ldots \subseteq \mathrm{SP}$
2. $\mathrm{FIN} \not\subseteq \mathrm{SP}$
3. For each k, SP_k is closed under intersection, but neither complement nor union.

(Rogers et al. 2010)

Characterizing Strictly Piecewise Languages

Subsequence Closure

A language L is Strictly Piecewise iff whenever $w \in L$ every subsequence of w also belongs to L.

Subsequence Closure helps us prove languages are not Strictly Piecewise.

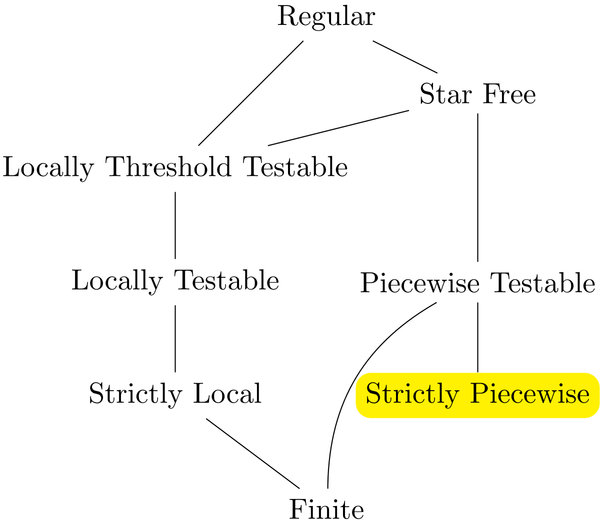
(Rogers et al. 2010)

Exercises

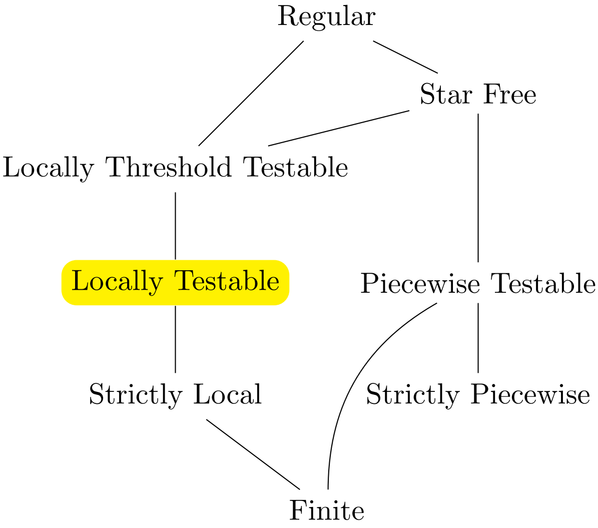
Prove the following languages are not SP.

1. Strings do not contain the substring `bb`.
2. Strings contain an odd number of `b`s.

Subregular Complexity



Subregular Complexity



Locally Testable Languages

A language L is Locally k-Testable if it is a Boolean combination of finitely many Strictly k-Local languages.

Boolean operators

- intersection $(L_1 \cap L_2$)u
- union $(L_1 \cup L_2)$
- complement $(\Sigma^* - L)$

A language is Locally Testable if it is Locally k-Testable for some k.

Locally Testable Languages

A language L is Locally k-Testable if and only if there is a set S of sets of k-factors such that

$$L = \{w \mid \texttt{factor}_k(\rtimes w \ltimes) \in S\}$$

Note: It follows a locally 2-testable language can include *aaba* and exclude *aa* but a strictly 2-local language cannot.

Grammar-independent characterization of Locally Testable Languages

Locally Testability

A language L is Locally Testable iff there exists k such that for all $u, v \in \Sigma^*$, if u and v have the same k-factors then $u, v \in L$ or $u, v \notin L$.

(McNaughton and Papert 1971, Rogers and Pullum 2011)

Exercise

Using Locally Testability, prove the language below is not LT.

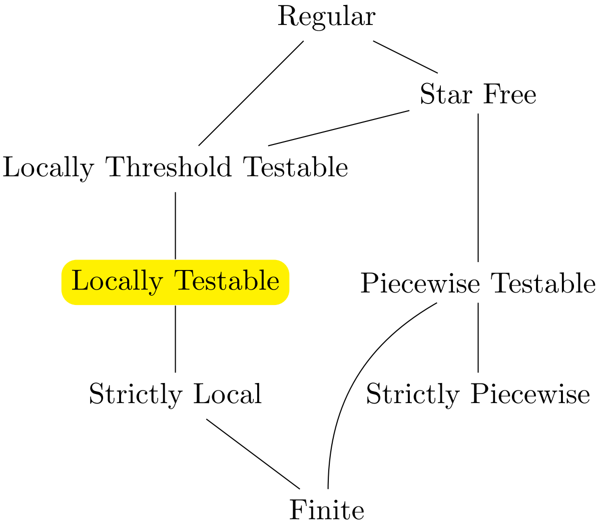
1. Strings contain exactly one `b`.
2. Strings can contain the subsequence sʃ but not ʃs.

Theorems

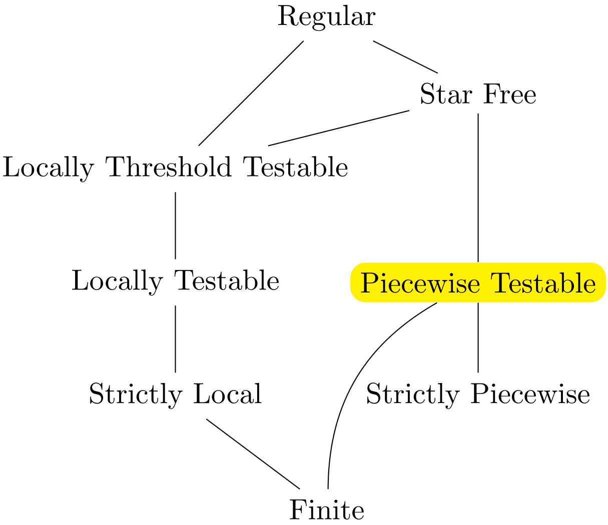
1. $\text{LT}1 \subsetneq \text{LT}2 \ldots \text{LT}k \subsetneq \text{LT}k+1 \ldots \subseteq \text{LT}$.
2. For each k, $\text{SL}k \subsetneq LTk$.
3. LT and SP are incomparable.
4. Closing LT under concatenation yields the Star-Free languages.

(McNaughton and Papert 1971, Rogers and Pullum 2011)

Subregular Complexity



Subregular Complexity



Piecewise Testable Languages

A language L is Piecewise k-Testable if it a Boolean combination of finitely many Strictly k-Piecewise languages.

Boolean operations (and regular expression equivalents)

- intersection (L_1 & L_2)
- union ($L_1 + L_2$)
- complement ($\overline{L}$)

A language is Piecewise Testable if it is Piecewise k-Testable for some k.

Piecewise Testable Languages

A language L is Piecewise k-Testable if and only if there is a set S of sets of k-subsequences such that

$$L = \{w \mid \texttt{subseq}_k(w) \in S\}$$

Note: It follows a piecewise 2-testable language can include *aaba* and exclude *aa* but a strictly 2-piecewise language cannot.

Grammar-independent characterization of Piecewise Testable Languages

Piecewise Testability

A language L is Piecewise Testable iff there exists k such that for all $u, v \in \Sigma^*$, if u and v have the same k-long subsequences then either $u, v \in L$ or $u, v \notin L$.

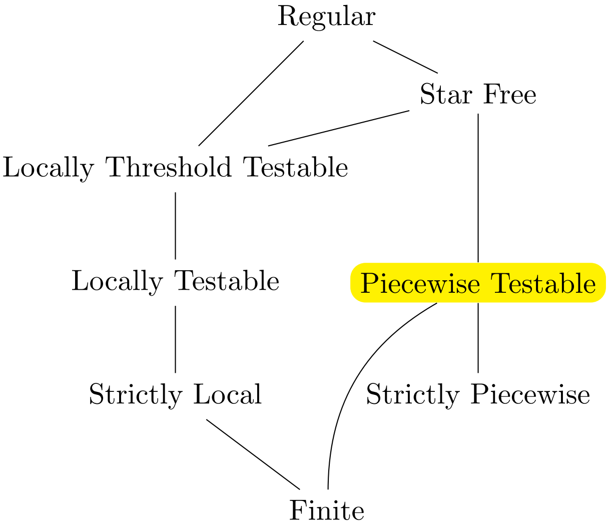
(Simon 1975)

Theorems

1. $\mathrm{PT}1 \subsetneq \mathrm{PT}2 \ldots \mathrm{PT}k \subsetneq \mathrm{PT}k+1 \ldots \subseteq \mathrm{PT}$.
2. For each k, $\mathrm{SP}k \subsetneq PTk$.
3. PT and LT are incomparable.

(Rogers et al. 2010, 2013)

Subregular Complexity



Subregular Complexity